Java and JavaScript

It is a common misconception that Java and JavaScript are just part of the same language. This is far from true. Although they are similarly named, there are quite a few differences between them, the first of which is their origin. Around June of 1991, Java was developed at Sun Microsystems, and was originally called Oak. It was officially announced (after much development and a name change) in May 1995 at SunWorld '95. JavaScript was developed at Netscape Communications Corporation and was originally called LiveScript. Sun renamed Oak to Java because of copyright issues with another language already called Oak, and Netscape changed LiveScript to JavaScript after an agreement with Sun to develop JavaScript as a language for "non-programmers." JavaScript was first released with Netscape 2.0.

JavaScript	Java
Developed by Netscape.	Developed by Sun.
Code is interpreted by client(Web browser).	Code is compiled and placed on server before execution on client.
Object-based. Objects are built in and extensible but are not classes and cannot use inheritance.	Object-oriented. Everything is a class that can use inheritance.
Data types need not be declared (loose typing).	Data types must be declared (strong typing).
Runtime check of object references (dynamic binding).	Compile-time check of object references (static binding).
Restricted disk access (must ask before writing a file).	Restricted disk access (levels of access set by user; cannot automatically write to disk).
Scripts are limited to Web browser functionality.	Compiled code can run either as a Web applet or a stand-alone application.
Scripts work with HTML elements (tags).	Can handle many kinds of elements (such as audio and video).
The language is rapidly evolving and changing in functionality.	Most major changes are complete.
There are few libraries of standard code libraries bundled with the language.	Java comes with many code examples and with with which to build Web applications.

JavaScript Is Not Java

The first thing you need to know is that JavaScript is *not* Java. You can write JavaScript scripts and never use a single Java applet. The reason JavaScript has adopted their name is because the language has a similar syntax to Java. Netscape also recognized the momentum building behind Java and leveraged the name to strengthen JavaScript. If you have programmed in Java, then you will find that JavaScript is both intuitive and easy for you to pick up. There will not be very much new for you to learn.

Interpreted versus Compiled

JavaScript code is almost always placed within the HTML document which it will be running. When you load a page that contains JavaScript code, the Web browser contains a built-in interpreter that takes the code as it is loaded and executes the instructions.

JavaScript uses the <script>...</script> tag, similar to Java's <applet>...</applet> tag. Everything within the <script>...</script> is ignored by Netscape's HTML parser, but is passed on to the JavaScript interpreter. For Web browsers that do not support the <script>...</script> tag, it is customary to further enclose the JavaScript code in comments.

Object-Based versus Object-Oriented

JavaScript takes liberally from Java in respect to its overall language structure, but lacks many of the features that make Java an object-oriented language. JavaScript has built-in objects (such as Navigator, Window, or Date) that access many browser elements such as windows, links, and images. Java typically cannot access any of these browser elements and is restricted to the area that contains it and any Java windows it subsequently creates.

JavaScript allows you to create new "objects" that are really functions. Objects in JavaScript are not true objects because they do not implement inheritance and other features that Java does. For instance, you cannot create a new class MyWindow that inherits properties of the JavaScript object window (the top-level object in JavaScript).

Java code can be written to allow a programmer to do just about anything conceivable on a computer as a stand-alone application or a Java applet. But, JavaScript fills a major gap (at least in the context of Java applets and Web browsers) by acting as a "glue" by which Java and the browser can communicate. Via JavaScript, you could enter information into a form field in an HTML document, and a Java applet on that page could use that input to display new information. JavaScript allows Java applets to gain access to properties of an HTML page, and allows non-programmers access to various parts of a Java applet--such as public variables.

Strong Typing versus Loose Typing

When you are writing JavaScript code, variables don't need to have data types when they are declared. This *loose typing* means that it is much easier for JavaScript writers to work on creating and manipulating variables without worrying if the data was an int, float, and so on.

In Java, you must explicitly declare what data type a variable will be before you use it. This is called *strong typing* and contributes to the overall stability of Java code, but can cause problems for new programmers. JavaScript allows you to ignore this and assumes that the type of value you first assign to a variable is the type you intended it to be. For instance, if you had a variable HouseType and you assigned it a value of "Victorian," JavaScript assumes you meant HouseType to be a string all along and does not complain if you did not specifically set HouseType to a string. However if you had assigned a value of 4 to HouseType, JavaScript now assumes it is of type INT.

Dynamic versus Static Binding

Because JavaScript is interpreted on the client's browser, object references are checked on-the-fly as opposed to Java's static binding at compile time. *Binding* simply means that a variable name is bound to a type-either statically through an explicit declaration of a type with a variable, or dynamically through an implicit association determined by the computer at compile or runtime. Because of JavaScript's dynamic nature, objects in JavaScript can be created on-the-fly as well, and might change the functionality of the script due to some outside factor (time, customer responses, and so on). The Date object is often used to get information about the Web browser's current date, time, day of the year, and more. This object is created at the time the JavaScript code is interpreted in order to get the correct information.

Java programs-with static binding-are typically more stable, because the entire process of compiling the code has already been completed via the Java compiler. Any bad or missing object references have been corrected. This is an advantage when you want the given application to load and run quickly on the user's machine.

Restricted Disk Access

Security is a hot issue in today's Internet and intranet Web industry for many good reasons. One of the greatest fears people have when they use the Web is that a hostile program will enter their computer and damage or compromise their sensitive data. Java has a comprehensive way of dealing with security that allows it to do useful things on your computer while keeping it isolated from your sensitive documents. Java applets can't write to your hard drive at all, or if they do, it is in some extremely limited way.

Because JavaScript can control so many aspects of your browser, for a while there was some concern that JavaScript was less safe than Java. This was primarily due to bugs in early versions of JavaScript that allowed it to send the contents of the file containing your bookmarks and e-mail address to another remote site through a hidden form, or acquire a list of all the files on your machine.

Note that JavaScript *can* write to your hard drive-an essential feature that your Web browser has in order to write to its cache or save files downloaded via the Web. However, it requires now that you specifically click Accept in a dialog box to download a file. In a sense, JavaScript is more versatile than Java in this respect, because it allows you to use your browser to create files to save and work with at a later time.

If you are concerned that some kind of hostile code might damage your machine, you should be aware that the possibility of virus infection has been around for a long time. Java, with its assertion of security, has only focused more attention to security issues. Soon, with JavaScript's *tainting* (a system of marking data so that it cannot be sent via a form or mailto: link via JavaScript) and Java's *digital signatures* (an electronic verification of the origin or identity of the code or information), you will be able to verify that any code–be it Java or JavaScript--comes from some trusted source. This is very good, because you will be able to allow Java to perform more sensitive tasks such as update your Oracle database, or send and auto-install updated versions of software on your machine.

Also note that your Web browser (and JavaScript through the browser) can also write information to a file called a *cookie*. This is a file on your machine that allows Web sites to store information about you that could be retrieved when you visit that site again. Only the site that wrote the information to the cookie can retrieve it, and it is usually only used to maintain some kind of temporary information about you to carry across different pages (such as a user ID, or the fact that you are from Florida).

Different Functionality (Scope Limitations) and Code Integration with HTML

JavaScript is limited in scope to your Web browser. Java, on the other hand, can run as a stand-alone application (like Microsoft Word) or within the context of a browser as an applet. It is a testimony of the versatility of Java that it has adapted so quickly to the Web. It was originally intended to run as operating system and controls on set-top boxes or other small communication appliances. Given that Java will eventually outstrip C++ as the programming language of choice, it has the functionality and versatility to run in many different operating systems.

JavaScript is a smaller language for a more limited audience of Web browser programmers. JavaScript gives to Web programmers the capability to access and modify all of the HTML tags, form elements, window elements, images, bookmarks, links, and anchors in a Web browser. It enables the programmer to create Web sites that respond and change based on many factors such as the time of day, or some user profile.

Libraries

Sun delivers Java with a standard set of libraries that act to dramatically enhance its usefulness. Instead of having to write all the code to handle images, sockets, and so on, the programmers at Sun have done this for you. You simply have to learn the standard APIs so you can quickly write terminal emulators, word processors, and more.

JavaScript-because of its relative youth-has not had time to build up any assemblage of standard code with which to build Web-based applications. One major problem that had stalled this development was that you were forced to embed your code in the HTML document in which you wanted to use the script. With the addition of the SRC attribute to the <script> tag, you can now write your code in a separate file and merely reference the script in the page. It is similar to the CODE attribute in the <applet> tag (this page would load all of the JS files, display the correct title at the top of the window, and display a clock above the Welcome to my home page text).

Use JavaScript to Write Text

JavaScript runs in the browser by being added directly into an existing HTML document. You can add special tags and commands to the HTML code that will tell the browser that it needs to run a script. Once the browser sees these, it interprets the JavaScript commands and will do what you have directed it to do with your code. Thus, by simply editing an HTML document, you can begin using JavaScript on your Web pages and see the results. For example, the following code adds some JavaScript to an HTML file that writes some text onto the Web page. Notice the addition of <SCRIPT> and </SCRIPT> tags. The code within them is JavaScript:

```
<HTML>
<BODY>
<SCRIPT language="JavaScript">
   document.write("<B>This text was written with JavaScript.</B>");
</SCRIPT>
</BODY>
</HTML>
```

Hiding JavaScript Code from Older Browsers

Web browsers that support JavaScript will execute the JavaScript statements that appear between the <SCRIPT> and </SCRIPT> tags. Browsers that don't support JavaScript, but that recognize the <SCRIPT> tag, will simply ignore everything between <SCRIPT> and </SCRIPT>. This is as it should be. Other, older browsers, however (and there are a lot of them), do not recognize the <SCRIPT> and </SCRIPT> tags, and so they ignore the tags themselves, and treat all the JavaScript between them as text to be displayed. Users of old browsers cannot run your JavaScript programs, and this should be punishment enough--they should not also have to look at your code!

In order to prevent this, you enclose the body of your scripts within an HTML comment

```
<SCRIPT LANGUAGE="JavaScript">
  <!-- begin HTML comment that hides the script
     .
     . JavaScript statements go here
     .
  // end HTML comment that hides the script -->
</SCRIPT>
```

Using External JavaScript Files

Using JavaScript on your pages? Style sheets too? When you put complex JavaScript code and style specifications into the HEAD section, you may end up with more lines of code there than actual content in the BODY section! Use external files to make your pages load faster, reduce coding errors, and increase search engine appeal.

Many good CSS and HTML techniques seem to require developers to dodge the guidelines recommended by the World Wide Web Consortium (W3C). Not for fun, but because many browsers don't completely support the W3C specifications.

Other external file advantages include:

- Smaller pages: Use external files to replace many lines of JavaScript or style definitions with a single line of code that calls the external file. Less code equals faster loading pages - and happier visitors.
- Easier maintenance: Easily include the same code or formatting information on every page. Need to change a font-family, color, or variable name? Just change it once in the external file and you see the change on every page.
- Search engine friendly: Search engine spiders like to see important content at the top of the page. Some even rank earlier content as more important. Pages with important content close to the top of the page are more spider-friendly.

External JavaScript files are a great way to use the same JavaScript functions on every page and reduce space and maintenance problems.

Some webmasters think they can also use external JavaScript files to hide their source code because the code doesn't display when visitors use the View Source option. That might dissuade some beginning users, but more the experienced ones will simply retrieve them from their browser cache files.

Remember to validate your JavaScript code before you put it into the external file. Once the code works, create your external file like this:

1. Open a text editor such as Notepad.
2. Either cut and paste your existing JavaScript code or create new code. Don't include opening and closing SCRIPT tags inside the external file.
3. Save the file in text-only format with a .js extension.
4. Include the file in the HEAD section of your Web pages. You can include more than one external JavaScript file on the page.

```
<script src="myscript1.js" language="javascript" type="text/javascript">
</script>

<script src="myscript2.js" language="javascript" type="text/javascript">
</script>
```

JavaScript Variables and Type Conversion Functions

A variable is an amount of memory space reserved to store a piece of information of your code. To use such a memory space, you must first let the browser know that you would need it. For the browser to reserve such a memory space for you, and to use it eventually, you must give it a name. The memory space that is reserved will be used to receive some values from your application and to make such values available when your code needs them. The content of this reserved memory space can change or vary regularly. For example, if you create a web page that acts as an employment application and include a text box for the first name, the content of the first name text box will be different depending on each visitor. The memory space that stores the first name of the visitors will vary from one visitor to another visitor. For this reason, it is called a variable.

The name of the variable:

- Must start with a letter or an underscore character
- After the first character, can contain letters, digits, and underscores
- Cannot contain space

Types of variables include:

- Integers - Can be expressed in decimal, octal, or hexadecimal form. When led by a 0x with characters from 0–9, and a–f, the value is hexadecimal such as "0x24". The equivalent octal value is 044 and decimal is 36.
- Floating point - Examples are 3.14, -3.14, 314e-2
- Boolean values (true or false).
- String values - May be enclosed by single ' or double " quotes. The \' or \"sequence of characters will insert a quote character into a string.
- Arrays of any of the above types.
- Objects.

The type if the variable is determined at its creation time, by the type of variable assigned to it. Therefore the example:

var firstvar=3

creates an integer variable. If the variable value is later changed to:

firstvar="Three"

It becomes a string. Note here that when the variable is created, the keyword "var" is used to create it. When referencing the variable, after its creation, or changing its value, the keyword "var" is not used.

Variable evaluation

When evaluating expressions, variable types are favored in the following order:

- Strings
- Floating point
- Integer
- Boolean

Type Conversion Functions

isNaN( ):The isNaN() function is used to check if a value is not a number.

```
document.write(isNaN(123));
```

eval( ): Converts a string to integer or float value. It can also evaluate expressions included with a string. Example:

```
value1 = eval("124/2");   //becomes 62.
```

parseInt(): Converts a string to an integer returning the first integer encountered which is contained in the string. If no integer value were found such as in the string "abcd", then a value of 0 is returned. In the example, value1 becomes 12.

```
value1 = parseInt("12b13") ,
```

parseFloat(): Returns floating point numbers the same as the parseInt function, but looks for floating point strings and returns a float.

`document.write(" " + parseFloat("50"))`	`50`
`document.write(" " + parseFloat("50.12345"))`	`50.12345`
`document.write(" " + parseFloat("32.00000000"))`	`32.00000000`
`document.write(" " + parseFloat("71.348  92.218  95.405"))`	`71.348`
`document.write(" " + parseFloat("37 aardvarks"))`	`37`
`document.write(" " + parseFloat("Awarded the best wine of 1999"))`	`NaN`

toString(): converts a Boolean value to a string and returns the result.

```
<script type="text/javascript">
   var boo = new Boolean(true)
   document.write(boo.toString())
</script>
```

typeof(): This function returns the type of the object it operates on. Values returned are string values and may be one of "undefined", "object", "function", "number", "Boolean", or "string". The example will return the string "number".

```
document.write(typeof(10))
```

valueOf():returns the number as a string. The string output is always equal to the number as it's represented in base 10.

```
NumberObject.valueOf()
```

JavaScript Functions
A function is a reusable code-block that will be executed by an event, or when the function is called.

To keep the browser from executing a script when the page loads, you can put your script into a function. A function contains code that will be executed by an event or by a call to that function. You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

```
<html>
<head>
   <script type="text/javascript">
      function displaymessage()
      {
         alert("Hello World!")
      }
   </script>
</head>
<body>
   <form>
      <input type="button" value="Click me!" onclick="displaymessage()" >
   </form>
</body>
</html>
```

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before the user hits the button. We have added an onClick event to the button that will execute the function displaymessage() when the button is clicked.

How to Define a Function
The syntax for creating a function is:

```
function functionname(var1,var2,...,varX)
{
   some code
}
```

var1, var2, etc are variables or values passed into the function. The { and the } defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name:

```
function functionname()
{
some code
}
```

Note: Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

The return Statement
The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement.

Example
The function below should return the product of two numbers (a and b):

```
function prod(a,b)
{
   x=a*b
   return x
}
```

When you call the function above, you must pass along two parameters:
product = prod(2,3)

The returned value from the prod() function is 6, and it will be stored in the variable called product.

JavaScript Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear "
```

If the variable visitor is equal to PRES, then put the string "Dear President " in the variable named greeting. If the variable visitor is not equal to PRES, then put the string "Dear " into the variable named greeting.

Conditional Statements

The conditional statements available in JavaScript are if ... else and switch. The general structure of the if ... else is shown below:

```
if (condition){
     truestatements
}
else {
     falsestatements
}
```

The true statements are executed if the condition is true. The false statements are executed if the condition is false.

The switch statement is also used as a conditional statement. It has the following syntax:

```
switch (expression){
     case label1:
       statement;
       break;
     case label2:
       statement;
       break;
     ...
     default: statement;
}
```

The value of the expression is evaluated, and if label matches this value, the label's statements are executed. If no matching label is found, the default statements are executed. The break after each set of statements is optional, but they ensure that the loop is exited after the statments associated with a certain label are executed.

Loops

The for and while statements are the two loop statements available to JavaScript. The for statement has the following general structure:

```
for ([initial expression]; [condition]; [increment expression]){
     statements
}
```

The intial expression is the expression of the initial condition of the loop. It usually contains an expression stating the intial value of a single variable. While the condition is true, the loop statements are executed. The increment expression describes how the variable is incremented each pass through the loop.

The while statement has the general structure:

```
while (condition) {
     statements
}
```

The statements in the while loop are executed when the condition is true.

There is also a do...while statement which looks like:

```
do { statements
} while (condition)
```

This loop statement executes once before the condition is checked. The loop executes until the condition is false.

Loop Examples

The following is an example utilizing some of the concepts already discussed. The following code:

```
<SCRIPT LANGUAGE="JavaScript1.1">
for (var i=1; i==50; i++ ){
     if ((i % 10) == 0) {
       document.write(i + "<BR>");
     }
     else {
       document.write(i + " ");
     }
}
</SCRIPT>
```

would result in:

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

```
<SCRIPT LANGUAGE="JavaScript1.1">
i = 0
while (i < 50)
{
     i++
     if ((i % 10) == 0) {
       document.write(i + "<BR>");
     }
     else {
       document.write(i + " ");
     }
}
</SCRIPT>
```

producing output:

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

Event Handlers
An event is something that happens, e.g. a mouse click on a button, an image that has loaded. Events usually occur as a result of human interaction with the browser, e.g. selecting a document to load, entering form information.

Event handlers are JavaScript methods, i.e. functions of objects, that allow us as JavaScript programmers to control what happens when events occur.
For example, the following form has an onSubmit event handler that displays an alert message:

```
<FORM NAME="formName" onSubmit="alert('Form submitted')">
<INPUT TYPE="SUBMIT">
</FORM>
```

The onSubmit event handler can be invoked in two instances, by the user pressing the submit button, or by another JavaScript submitting the form, e.g.

```
document.formName.submit();
```

JavaScript event handlers have 'on' before the event name, e.g. onClick, onSubmit, onLoad, onFocus.
In JavaScript 1.1 it is possible to retrieve the event handlers contents, but the whole event handler must be in lower case, e.g. onsubmit:

```
document.write(document.formName.onsubmit);
```

It is also possible to set the contents of the event handler:

```
document.formName.onsubmit = 'function onsubmit(event) { alert("Changed"); }' ;
```

The complete list of Event Handlers:

- onAbort - The user aborts the loading of an image
- onBlur - form element loses focus or when a window or frame loses focus.
- onChange - select, text, or textarea field loses focus and its value has been modified.
- onClick - object on a form is clicked.
- onDblClick - user double-clicks a form element or a link.
- onDragDrop - user drops an object (e.g. file) onto the browser window.
- onError - loading of a document or image causes an error.
- onFocus - window, frame, frameset or form element receives focus.
- onKeyDown - user depresses a key.
- onKeyPress - user presses or holds down a key.
- onKeyUp - user releases a key.
- onLoad - browser finishes loading a window or all of the frames within a frameset.
- onMouseDown - user depresses a mouse button.
- onMouseMove - user moves the cursor.
- onMouseOut - cursor leaves an area or link.
- onMouseOver - cursor moves over an object or area.
- onMouseUp - user releases a mouse button.
- onMove - user or script moves a window or frame.
- onReset - user resets a form.
- onResize - user or script resizes a window or frame.
- onSelect - user selects some of the text within a text or textarea field.
- onSubmit - user submits a form.
- onUnload - user exits a document.

JavaScript 1.2
JavaScript 1.2 is supported by Netscape Navigator 4.0. Although Microsoft Internet Explorer appears to support JavaScript 1.2, it actually implements a different Dynamic Object Model which will not be descibed here.

Object	Event Handlers
area	onDblClick(), onMouseOut(), onMouseOver()
button	onBlur(), onClick(), onFocus(), onMousedown(), onMouseup()
checkbox	onBlur(), onClick(), onFocus()
document	onClick(), onDblClick(), onKeyDown(), onKeyPress(), onKeyUp(), onMouseDown(), onMouseUp()
fileupload	onBlur(), onChange(), onFocus()
form	onReset(), onSubmit()
frame	onBlur(), onDragDrop(), onError(), onFocus(), onLoad(), onMove(), onResize(), onUnload()
image	onAbort(), onError(), onKeyDown(), onKeyPress(), onKeyUp(), onLoad()
layer	onMouseOver(), onMouseOut(), onLoad(), onFocus(), onBlur()
link	onClick(), onDblClick(), onKeyDown(), onKeyPress(), onKeyUp(), onMouseDown(), onMouseOut(), onMouseUp(), onMouseOver()
password	onBlur(), onFocus()
radio	onBlur(), onClick(), onFocus()
reset	onBlur(), onClick(), onFocus()
select	onBlur(), onChange(), onFocus()
submit	onBlur(), onClick(), onFocus()
text	onBlur(), onChange(), onFocus(), onSelect()
textarea	onBlur(), onChange(), onFocus(), onKeyDown(), onKeyPress(), onKeyUp(), onSelect()
window	onBlur(), onDragDrop(), onError(), onFocus(), onLoad(), onMove(), onResize() onUnload()

Objects

Creating an object requires two steps:

- First, declare the object by using an object function
- Lastly, instantiate the newly created object by using the "new" keyword

Lets take this one step at a time. We will now proceed to create an object called "userobject", which, at this stage, does nothing:

Step 1: declare the object by using an object function

The first step towards creating an object requires us to define an object function. An object function is virtually identical in syntax as a regular function, although there are some differences which will surface later on. The object function is used to define and declare an object:

```
function userobject(parameter){
}
```

The parameter is optional, and with it, allows us to pass in values to an object. For example, in the pre-built object window.alert, the parameter is the text passed in to be alerted. Now, with just the above object function, we have in essence just created a new object called "userobject"! It does nothing at this stage, and will continue to do until we add in properties and methods. To use this object, all we have to do is instantiate it, by using the keyword "new".

Step 2: Instantiate the newly created object by using the "new" keyword

Once we've defined an object function, we have to instantiate it to actually use it. Instantiating an object function means using the keyword "new" in front of the object name, and then creating an instance of the object by assigning it to a variable:

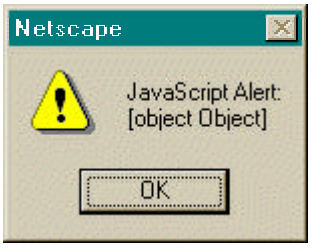
```
<script type="text/javascript">
function userobject(parameter){
}
//myobject is now an object of type userobject!
var myobject=new userobject("hi")
</script>
```

"myobject" is now an object...an instance of "userobject", to be exact. If you're a little confused at this stage, consider a more familiar example:

```
var image1=new Image(20,20)
```

The above should be review to us; we created an instance of the pre-built image object by assigning it to the variable image1. Well, this familiar process is exactly what we'll doing with the custom object above. If you're the kind that need to actually see and touch an object before you believe its an object, the window.alert method can help:

```
<script type="text/javascript">
function userobject(parameter){
}
//myobject is now an object of type userobject!
var myobject=new userobject("hi")
alert(myobject)
</script>
```



How to add properties to your own object

Thus far, our object "userobject" cannot do anything but take up space in a document. With some properties, that should all change. To add properties to a user defined object, directly embed the properties into the object function, with each property proceeded by the keyword "this" plus dot (.): In the below example, we'll extend "userobject" to contain two properties, each containing a string of text:

```
function userobject(parameter){
this.firstproperty=parameter
this.secondproperty="This is the second property"
}
```

Now, to use these properties, simply access them like accessing any other property:

```
<script>
var myobject=new userobject("hi there.")
//alerts "hi there."
alert(myobject.firstproperty)
//writes "This is the second property"
document.write(myobject.secondproperty)
</script>
```

How to add methods to your own object

Adding methods to a user defined object is a bit more complicated. We need to first declare and define a function for each method, then associate this function with the object function. For the sake of simplicity, we will simply call functions defined for methods "method functions." Lets get a clean start, and create a new object called "circle" that will contain methods that compute the area and diameter of a circle, respectively.

The first step to adding methods is to implement the method functions. Method functions define what a method does:

```
//first method function
function computearea(){
   var area=this.radius*this.radius*3.14
   return area
}

//second method function
function computediameter(){
   var diameter=this.radius*2
   return diameter
}
```

In the above case, we've created two method functions, "computearea" and "computediamter", which calculates various aspects of a circle. The two functions, as you can see, are just functions, with one major distinction. Take the first one, for example:

```
function computearea(){
   var area=this.radius*this.radius*3.14
   return area
}
```

What is this.radius? It looks like a property of a custom object. Since a method function will eventually be connected to the custom object, it has access to the properties of the object. We haven't defined the properties yet, but we will, and the method functions will use them in its calculation.

We will now associate the two method functions above to the new object "circle", so they become methods of the object:

```
<script type="text/javascript">
/*the below creates a new object, and gives it the two methods defined earlier*/
function circle(r){
   //property that stores the radius
   this.radius=r
   this.area=computearea
   this.diameter=computediameter
}
</script>
```

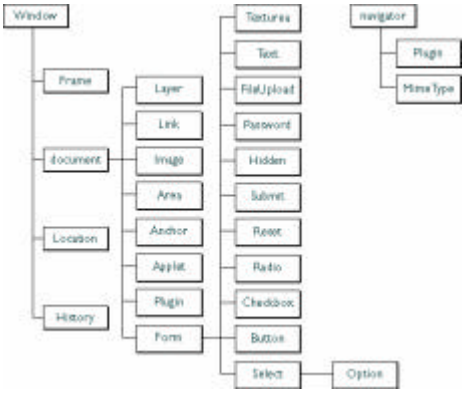
Finally, to use these methods, instantiate the object, and access the methods just like any other method:

```
<script type="text/javascript">
var mycircle=new circle(20)
//alerts 1256
alert("area="+mycircle.area())
//alerts 400
alert("diameter="+mycircle.diameter())
</script>
```

Document Object Model

Often referred to as the DOM, this object model is a hierarchy of all objects "built in" to JavaScript. Most of these objects are directly related to characteristics of the Web page or browser. The reason we qualify the term "built in" is because the DOM is technically separate from JavaScript itself. That is, the JavaScript language specification, standardized by the ECMA, does not actually specify the nature or specifics of the DOM. Consequently, Netscape and Microsoft have developed their own individual DOM's which are not entirely compatible. Additionally, the DOM stands apart from JavaScript because it could theoretically be accessed by other scripting languages as well. In summary, then, what we refer to as "JavaScript" is actually made up of both JavaScript, the language, and the DOM, or object model which JavaScript can access.

Below is a graphical chart illustrating a high-level view of Netscape's DOM. Microsoft's DOM is actually a superset of Netscape's, and so the chart below actually represents a subset of Microsoft's own DOM.



Navigator object

The navigator object was conceived back in the days when Netscape Navigator reined supreme. These days it serves as much as an irony of NS's diminished market share as way of probing browser information.

The navigator object of JavaScript contains the following core properties:

Properties	Description
appCodeName	The code name of the browser.
appName	The name of the browser (ie: Microsoft Internet Explorer).
appVersion	Version information for the browser (ie: 4.75 [en] (Win98; U)).
cookieEnabled	Boolean that indicates whether the browser has cookies enabled. IE4 and NS6+.
language	Returns the default language of the browser version (ie: en-US). **NS4 and NS6+ only.**
mimeTypes[]	An array of all MIME types supported by the client. **NS4 and NS6+ only.** Array is always empty in IE.
platform[]	The platform of the client's computer (ie: Win32).
plugins	An array of all plug-ins currently installed on the client. **NS4 and NS6+ only.** Array is always empty in IE.
systemLanguage	IE4+ property that returns the default language of the operating system. Similar to NS's language property.
userAgent	String passed by browser as user-agent header. (ie: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1))
userLanguage	IE4+ property that returns the preferred language setting of the user. Similar to NS's language property.

Let's see exactly what these properties reveal of *your* browser:

AppCodeName: Mozilla
AppName: Microsoft Internet Explorer
AppVersion: 4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)
UserAgent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)
Platform: Win32

At a glance

You probably think you have a very solid idea now of how to utilize the navigator object to detect your client's browser type. At its most basic form, the following two properties are used:

```
navigator.appName
navigator.appVersion
```

For example:

```
//detect Netscape 4.7+
if (navigator.appName=="Netscape"&&parseFloat(navigator.appVersion)>=4.7)
alert("You are using Netscape 4.7+")
```

However, depending on the browser you're trying to detect, you'll find the above two properties too limiting.

Detecting Firefox 1.0+

Take for example, Firefox 1.0+. It shares the same "appName" value as older Netscape browsers, which is "Netscape." The appVersion value returned is also out of wack, which is "5." So we need to look to another property, which turns out to be "UserAgent." For Firefox 1.04 for example, its userAgent property reads:

UserAgent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.8) Gecko/20050511 Firefox/1.0.4
Text at the end of the string apparently contains the information we want. Based on this, the below code detects if you're using Firefox 1.0+:

```
if(navigator.userAgent.indexOf("Firefox")!=-1){
   var versionindex=navigator.userAgent.indexOf("Firefox")+8
if (parseInt(navigator.userAgent.charAt(versionindex))>=1)
   alert("You are using Firefox 1.x or above")
}
```

The code assumes that all Firefox versions contain the string "Firefox/" immediately followed by the version number (ie "1").

Detecting IE 5.5+

Detecting IE using the navigator object also poses a similar problem as Firefox, since its "navigator.appVersion" property contains more than just the version number of IE:

```
4.0 (compatible; MSIE 5.5; Windows 98; Hotbar 3.0)
```

If we were to use parseFloat to try and extract out the browser version in "navigator.appVersion", we'd get 4.0, instead of the correct number (5.5). This is due to the way parseFloat works- by returning the first number it encounters. So, how does one go about getting the version number in IE? As with Firefox, it really comes down to how you wish to approach the issue. Lets stick with using the "navigator.appVersion" property this time, though you can certainly use "navigator.userAgent" as well:

```
//Detect IE5.5+
version=0
if (navigator.appVersion.indexOf("MSIE")!=-1){
   temp=navigator.appVersion.split("MSIE")
   version=parseFloat(temp[1])
}
if (version>=5.5) //NON IE browser will return 0
   alert("You're using IE5.5+")
```

In the above, string.split() is first used to divvy up navigator.appVersion into two parts, using "MSIE" as the separator:
4.0 (compatible; MSIE 5.5; Windows 98; Hotbar 3.0)
As a result temp[1] contains the part after "MSIE", with the browser version appearing first. Doing a parseFloat() on temp[1] therefore retrieves the browser version of IE.

Document Object

Document is the parent object of numerous other objects, such as "images", "forms" etc.
Note: "[]" surrounding a parameter below means the parameter is optional.

Properties	Description
alinkColor	Specifies the color of activated links in the document (Alink attribute).
all[]	IE4+ exclusive array that contains all of the elements within the document. Use document.all["elementID"] or document.all.elementID to access an element.
anchors[]	An array containing all of the anchors in the document.
applets[]	An array containing all of the applets in the document.
bgColor	Specifies the background color of the document.
cookie	A string containing the name/value pair of cookies in the document.
domain	Specifies the domain name of the server that served a document. Used for security purposes.
embeds[]	An array containing all of the plug-ins in the document, represented using the <embed> tag.
fgColor	Specifies the default text color of the document (text attribute).
fileSize	Returns the file size of the current document. In IE Windows, a numeric string is returned, while in IE Mac, a number instead. **IE only property.**
forms[]	An array containing all of the forms on the page.
images[]	An array containing all of the images on the page.
lastModified	Specifies the last modified date of the document, as reported by the web server.
linkColor	Specifies the color of unvisited links in the document (link attribute).
links[]	An array containing all of the links on the page.
plugins[]	Same as embeds[] object.
readyState	IE4+ exclusive property that specifies the loading status of the document. It returns one of the below 4 values: 1) uninitialized- The document hasn't started loading yet. 2) loading- The document is loading. 3) interactive- The document has loaded enough whereby user can interact with it. 4) complete- The document has fully loaded.
referrer	A string that specifies the URL in which the user derived from to reach the current, usually via a link.
title	Specifies the title of the document. Read/write in modern browsers.
URL	A string that specifies the complete URL of the document.
vlinkColor	Specifies the color of visited links in the document (vlink attribute).

Methods	Description
close()	Closes a document stream opened using document.open().
getElementById("ID")	A cross browser (IE5/NS6+) DOM method for accessing any element on the page via its ID attribute.
open([mineType])	Opens a document stream in preparation for document.write() to write to it. Use the optional "mineType" argument (default is text/html) to specify a specific minetype, such as "image/gif."
write("string")	Writes to the document (as it's loading) or document stream the "string" entered.
writeln("string")	Writes to the document (as it's loading) or document stream the "string" entered and inserts a newline character at the end.

Examples

open([mineType])
This example opens a blank window and writes to it:

```
win2=window.open("") //open blank window and write to it
win2.document.open() //open document stream
win2.document.write("<b>Some text</b>")
win2.document.close()
```

write("string")
Writes out the current document's URL while the page is loaded:

```
document.write("This page\'s URL is "+document.URL)
```

JavaScript Form Object

Forms on the page are represented in JavaScript via the form object, and can be accessed in one of the 3 standard ways below:

```
document.yourformname //where yourformname is the name of your form.
document.form.yourformname
document.forms["yourformname"]
```

Events	Description
onReset	Code is executed when the form is reset (by clicking on "reset" button)
onSubmit	Code is executed when form is submitted.

Properties	Description
action	Read/write string that reflects the action attribute of the form.
elements[]	An array containing all of the elements of the form. Use it to loop through form easily.
encoding	Read/write string that specifies how the form data is encoded.
length	The number of elements in the form.
method	Read/write string that specifies how the method the form is submitted.
name	The name of the form.
target	The name of the target frame or window form is to be submitted to.

Methods	Description
reset()	Resets the form.
submit()	Submits a form.

submit() example

```
<a href="javascript:document.myform.submit()"><img src="submit.gif" /></a>
```

Window Object
The Window object is the top level object in JavaScript, and contains in itself several other objects, such as "document", "history" etc.

Events	Description
onblur	Code is executed when the window loses focus.
onerror	Code is executed when a JavaScript error occurs.
onfocus	Code is executed when the focus is set on the current window.
onload	Code is executed when the page has finished loading.
onresize	Code is executed when the window is resized.
onunload	Code is executed when the page is unloaded (visitor leaves the page).

*Most of the events to the left can be defined in two ways. Using "onload" as an example:

```
<body onload="dothis()">
window.onload=dothis //inside your script
```

Properties	Description
closed	Returns the Boolean variable indicating whether window has been closed or not.
defaultStatus	Read/write property that reflects the default window status bar message that appears.
document	Reference to the current document object.
frames	An array referencing all of the frames in the current window. Use frames.length to probe the number of frames.
history	Reference to the History object of JavaScript, which contains information on the URLs the visitor has visited within the window.
length	Returns the number of frames contained in the window.
location	Reference to the Location object of JavaScript, which contains information on the current URL.
name	The name of the window as optionally specified when calling window.open().
opener	Contains a reference to the window that opened the secondary window via window.open(). This property should be invoked in the secondary window.
parent	Reference to the parent window of the current window, assuming current window is a frame. Otherwise, it simply refers to current window.
self	A synonym for the current window.
status	A read/write property that allows you to probe and write to the browser's status bar.
top	A synonym for the topmost browser window.
window	References the current window. Same as "self."
innerWidth, innerHeight	Specifies the width and height, in pixels, of the window's content area respectively. Read/Write. Does not include the toolbar, scrollbars etc. **NS4 and NS6+ only.** **Note:** IE4+ equivalents are "document.body.clientWidth" and "document.body.clientHeight"
outerWidth, outerHeight	Specifies the total width and height, in pixels, of the window's content area respectively. Read/Write. Includes any toolbar, scrollbars etc. **NS4 and NS6+ only.**
pageXOffset, pageYOffset	Returns an integer representing the pixels the current document has been scrolled from the upper left corner of the window, horizontally and vertically, respectively. **Note:** IE4+ equivalents are "document.body.scrollLeft" and "document.body.scrollTop"
screenX, screenY	Specifies the x and y coordinates of the window relative to the user's monitor screen. **NS4 and NS6+ only.**
screenLeft, screenTop	Specifies the x and y coordinates of the window relative to the user's monitor screen. **IE5+ only.**

Methods	Description
alert(msg)	Displays an Alert dialog box with the desired message and OK button.
blur()	Removes focus from the window in question, sending the window to the background on the user's desktop.
clearInterval(ID)	Clears the timer set using ID=setInterval().
clearTimeout(ID)	Clears the timer set using ID=setTimeout().
close()	Closes a window.
confirm(msg)	Displays a Confirm dialog box with the specified message and OK and Cancel buttons.
find(string, [casesensitive], [backward])	Searches for the "string" within the page, and returns string or false, accordingly. "casesensitive" is a Boolean denoting whether search is case sensitive. "backwards" is a Boolean when set to true, searches the page backwards. Final two optional parameters must be set together or none at all. **NS4/NS6+ exclusive method.**
focus()	Sets focus to the window, bringing it to the forefront on the desktop.
home()	Navigates the window to the homepage as designated by the user's browser setting. **NS4/NS6+ only.**
moveBy(dx, dy)	Moves a window by the specified amount in pixels.
moveTo(x, y)	Moves a window to the specified coordinate values, in pixels.
open(URL, [name], [features], [replace])	Opens a new browser window. "Name" argument specifies a name that you can use in the target attribute of your <a> tag. "Features" allows you to show/hide various aspects of the window interface. "Replace" is a Boolean argument that denotes whether the URL loaded into the new window should add to the window's history list. A value of true causes URL to not be added.
print()	Prints the contents of the window or frame.
prompt(msg, [input])	Displays a Prompt dialog box with a message. Optional "input" argument allows you to specify the default input (response) that gets entered into the dialog box. Set "input" to "" to create a blank input field.
resizeBy(dx, dy)	Resizes a window by the specified amount in pixels.
resizeTo(x y)	Resizes a window to the specified pixel values.
scrollBy(dx, dy)	Scrolls a window by the specified amount in pixels.
scrollTo(x, y)	Scrolls a window to the specified pixel values.
setInterval("func", interval, [args])	Calls the specified "func" (or a JavaScript statement) repeatedly per the "interval" interval, in milliseconds (ie: 1000=every 1 second). "func" must be surrounded in quotations, as if it was a string. Use the optional "args" to pass any number of arguments to the function.
setTimeout("func", interval)	Calls the specified "func" (or a JavaScript statement) once after "interval" has elapsed, in milliseconds (ie: 1000=after 1 second). "func" must be surrounded in quotations, as if it was a string.
stop()	Stops the window from loading. NS4/NS6+ exclusive method.

Window Features in window.open()
The below lists the string features you can pass into the "feature" parameter of window.open() to manipulate its interface. Most features support a value of true or false, though in general, simply including the name of the feature implies it should be added to the window (yes), while not including it means it shouldn't (no). Separate each feature with a comma (,).

Feature	Description
channelmode	Specifies if window should be opened in channel mode. IE only.
fullscreen	Specifies if window should be opened in full screen mode. IE only.
height	Specifies the height of the window.
left	Specifies the x coordinates of the window in pixels. IE only. See "screenX" as well.
location	Specifies if the location bar of the window should be included.
menubar	Specifies if the menu bar of the window should be included.
resizable	Specifies if window should be resizable.
screenX	Specifies the x coordinates of the window in pixels. NS only. See "left" as well.
screenY	Specifies the x coordinates of the window in pixels. NS only. See "top" as well.
scrollbars	Specifies if window should contain scrollbars.
status	Specifies if the status bar of the window should be included.
toolbar	Specifies if the toolbar of the window (ie: reload button) should be included.
top	Specifies the y coordinates of the window in pixels. IE only. See "screenY" as well.
width	Specifies the width of the window.

Examples
open(URL, [name], [features], [replace])

```
window.open("http://www.northeastern.edu", "", "width=800px, height=600px, resizable")
```

confirm(msg)

```
var yourstate=window.confirm("Are you sure you are ok?")
if (yourstate) //Boolean variable. Sets to true if user pressed "OK" versus "Cancel."
window.alert("Good!")
```

prompt(msg, [input])

```
var thename=window.prompt("please enter your name")
window.alert(thename)
```

Arrays

Arrays are variables, which can contain multiple values. The difference is that it is only "one" array, which will store these values. If you needed to store multiple values in normal variables, then you would have to define multiple variables. Usually arrays are associated with numbers, which starts with 0. If you create an array for 4 elements, then the first one will be 0, the second 1, the third 2 and so on.

There are three ways of defining an array:

```
var myfriends=new Array() //1) regular array. Pass an optional integer argument to control array's size.
myfriends[0]="John"
myfriends[1]="Bob"
myfriends[2]="Sue"
```

```
var myfriends=new Array("John", "Bob", "Sue") //2) condensed array
```

```
var myfriends=["John", "Bob", "Sue"] //3) literal array
```

Overview of declaring dense arrays

A dense array allows programmers to efficiently declare an array when the initial values of the array is known beforehand. Dense arrays are supported in all major browsers. But before we officially get to dense arrays, lets review the syntax for a regular array first.

There is an alternative way of assigning values to an array during declaration. The below illustrates this form:

```
var myarray=new Array("one","two","three")
```

The above is what's called a dense array, since it compresses the declaration and initialization of an array into one step. As you can see, we directly passed in the values of the array as parameters. To access the individual values inside, simply use the same old notation:

```
alert(myarray[0]) //alerts "one"
```

Example: Writing out the current date and time using dense arrays

A dense array excels over a normal one when it comes to initializing a rather large array with values upon declaration. In this section, we'll create a script that writes out the current date and time, in the following format:

```
Tuesday, July 28, 2005
```

The obvious task at hand is to first declare a series of variables that store the text "Monday", "Tuesday" etc and "January", "February", etc, since JavaScript is not human, and does not by default understand our calendar system. The remaining work is to match these text with the current date, and write it out. The tedious part is the definition of these text, and the perfect job for arrays to handle- dense arrays, to be precise:

```
var dayarray=new Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")

var montharray=new Array("January","February","March","April","May","June","July","August","September","October","November","December")
```

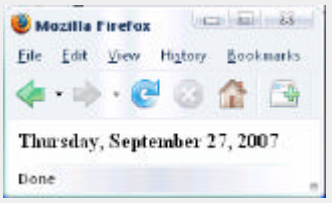
The above code not only efficiently stores the text for the days of the week and months into two arrays, but allows us to directly use the JavaScript prebuilt functions date.getDay() and date.getMonth() as the index to reference the current date as represented by text, and write them out. Let's see just how this is done. The below shows the complete code that outputs the current date and time:

```
<script type="text/javascript">
var mydate = new Date()
var year   = mydate.getFullYear()
var day    = mydate.getDay()
var month  = mydate.getMonth()
var daym   = mydate.getDate()

//if the current date is less than 10, pad it.
if (daym<10)
   daym = "0" + daym

var dayarray=new Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")
var montharray=new Array("January","February","March","April","May","June","July","August","September","October","November","December")

//write out the final results
document.write("<b>"+dayarray[day]+", "+montharray[month]+" "+daym+", "+year+"</b>")
</script>
```



Literal Notation Overview

Literal notation is a form of array declaration introduced in JavaScript 1.2. Like the language version, it's supported by 4th+ generations of both IE and NS. In other words, all modern browsers.

To declare a literal notation (array), start out with square brackets, then enclose each participating value inside, separated by commas:

```
var myarray=["Joe", "Bob", "Ken"]
```

Once declared, a literal notation behaves very much like a normal array, so to call to Joe, you would use:

```
alert(myarray[0])
```

At this point, we can all discern at least one merit of literal notation- its compact syntax .

Kinds of values literal notation accepts

Ok, moving on, lets explain the kind of values literal notation supports. Apart from the obvious "string" and "numeric" input, you can also use expressions:

```
var myarray=[x+y, 2, Math.round(z)]
```

If you can't make up your mind what to enter, undefined values are accepted too, by using a comma (') in place of the value::

```
var myarray=[0,,,,5]
```

In the above, there are actually 6 array values, except the ones in the center are all undefined. This is useful, for example, if you wish to come back later to dynamically fill in these values, or set up your array for future expansion.

JavaScript Math Object

The Math object allows you to perform common math related tasks. Here are a few example uses:

```
var mygrade = Math.round(8.6) //returns 9
var mynum   = Math.pow(2, 3)  //returns 8
```

Properties	Description
E	The constant of E, the base of natural logarithms.
LN2	The natural logarithm of 2.
LN10	The natural logarithm of 10.
LOG2E	Base 2 logarithm of E.
LOG10E	Base 10 logarithm of E.
PI	Returns PI.
SQRT1_2	Square root of 1/2.
SQRT2	Square root of 2.

Methods	Description
abs(x)	Returns absolute value of x.
acos(x)	Returns arc cosine of x in radians.
asin(x)	Returns arc sine of x in radians.
atan(x)	Returns arc tan of x in radians.
atan2(y, x)	Counterclockwise angle between x axis and point (x,y).
ceil(x)	Returns the smallest integer greater than or equal to x. (round up).
cos(x)	Returns cosine of x, where x is in radians.
exp(x)	Returns e^x
floor(x)	Returns the largest integer less than or equal to x. (round down)
log(x)	Returns the natural logarithm (base E) of x.
max(a, b)	Returns the larger of a and b.
min(a, b)	Returns the lesser of a and b.
pow(x, y)	Returns X^y
random()	Returns a pseudorandom number between 0 and 1. Example(s).
round(x)	Rounds x up or down to the nearest integer. It rounds .5 up. Example(s).
sin(x)	Returns the Sin of x, where x is in radians.
sqrt(x)	Returns the square root of x.
tan(x)	Returns the Tan of x, where x is in radians.

Examples:

random()

```
Math.random() //returns random number between 0 and 1, such as 0.634343434...
Math.floor(Math.random()*11) //returns random integer between 0 and 10
```

round(x)

```
Math.round(25.9)  //returns 26
Math.round(25.2)  //returns 25
Math.round(-2.58) //returns -3

var original=28.453
//round "original" to two decimals:
Math.round(original*100)/100 //returns 28.45
```

JavaScript Number Object

The Number object is an object wrapper for primitive numeric values. It contains some useful methods, introduced in JavaScript 1.5.

Constructor:

```
new Number(value) //"value" is the numeric value of the Number object to be created or value to be converted to a number.
Number (value)
```

Properties	Description
MAX_VALUE	The largest representable number in JavaScript.
MIN_VALUE	The smallest representable number in JavaScript.
NaN	"Not a number" value.
NEGATIVE_INFINITY	Negative infinity, returned on overflow.
POSITIVE_INFINITY	Infinity, returned on overflow.
prototype	Prototype property, to add custom properties and methods to this object.

Methods	Description
toExponential(x)	Returns a number in exponential notation (string format). "x" should be a number between 0 and 20, dictating the number of digits to include in the notation after the decimal place.
toFixed(x)	Formats any number for "x" number of trailing decimals. The number is rounded up, and "0"s are used after the decimal point if needed to create the desired decimal length.
toPrecision(x)	Formats any number so it is of "x" length. Also called significant digits. A decimal point and "0"s are used if needed to create the desired length.

Examples

toFixed(x)

```
var profits=2489.8237
profits.toFixed(3) //returns 2489.824 (round up)
profits.toFixed(2) //returns 2489.82
```

JavaScript Date Object

There are fours ways of instantiating a date object: (most parameters here are optional, if not specified, 0)

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Here are a few examples of instantiating a date:

```
today    = new Date()
birthday = new Date("March 11, 1985 09:25:00")
birthday = new Date(85,2,11)
birthday = new Date(85,2,11,9,25,0)
```

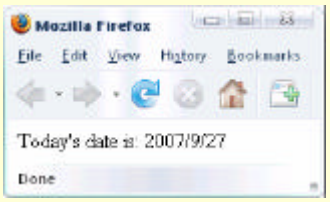
Methods	Description
getFullYear()	Returns year in full 4 digit format (ie: 2004).
getYear()	Returns the year, in 4 digit format or otherwise depending on browser. Deprecated in favor of getFullYear().
getMonth()	Returns the month. (Range is 0-11)!
getDate()	Returns the day of the month (Range is 1-31)
getDay()	Returns the day of the week (Range is 0-6). 0=Sunday, 1=Monday, etc.
getHours()	Returns the hour (Range is 0-23).
getMinutes()	Returns the minutes. (Range is 0-59).
getSeconds()	Returns the seconds. (Range is 0-59).
getMilliseconds()	Returns the milliseconds. (Range is 0-999).
getTime()	Returns the millisecond representation of the current Date object. In the words, the number of milliseconds between 1/1/1970 (GMT) and the current Date object.
getTimezoneOffset()	Returns the offset (time difference) between Greenwich Mean Time (GMT) and local time of Date object, in minutes.
getUTCFullYear()	Returns the full 4 digit year in Universal time.
getUTCMonth()	Returns the month in Universal time.
getUTCDate()	Returns the day of the month in Universal time.
getUTCDay()	Returns the day of the week in Universal time.
getUTCHours()	Returns the hour in Universal time.
getUTCMinutes()	Returns the minutes in Universal time.
getUTCSeconds()	Returns the seconds in Universal time.
getUTCMilliseconds()	Returns the milliseconds in Universal time.
setFullYear(year, [month], [day])	Sets the year of the Date object. (year: 4 digit year).
setYear(year)	Sets the year of the Date object. "year" can be two digits (1900 is automatically added to it), or 4 digits. Deprecated over setFullYear() above.
setMonth(month, [day])	Sets the month of the Date object. (month: 0-11)
setDate(day_of_month)	Sets the day of the month of the Date object. (day_of_month: 1-31).
setHours(hours, [minutes], [seconds], [millisec])	Sets the hour of the Date object. (hours: 0-23).
setMinutes(minutes, [seconds], [millisec])	Sets the minutes of the Date object. (minutes: 0-59).
setSeconds(seconds, [millisec])	Sets the seconds of the Date object. (seconds: 0-59).
setMilliseconds(milli)	Sets the milliseconds field of the Date object. (milli: 0-999)
setTime(milli)	Sets the value of the Date object in terms of milliseconds elapsed since 1/1/1970 GMT.
setUTCFullYear(year, [month], [day])	Sets the year of the Date object in Universal time.
setUTCMonth(month, [day])	Sets the month in Universal time.
setUTCDate(day_of_month)	Sets the day of the month in Universal time.
setUTCHours(hours, [minutes], [seconds], [millisec])	Sets the hours in Universal time.
setUTCMinutes(minutes, [seconds], [millisec])	Sets the minutes in Universal time.
setUTCSeconds(seconds, [millisec])	Sets the seconds in Universal time.
setUTCMilliseconds(milli)	Sets the milliseconds in Universal time.
toGMTString()	Converts a date to a string, using the GMT conventions. Deprecated in favor of toUTCString().
toLocaleString()	Converts a date to a string, using current locale conventions.
toLocaleDateString()	Returns the date portion of the Date as a string, using current locale conventions.
toLocaleTimeString()	Returns the time portion of the Date as a string, using current locale conventions.
toString()	Converts a Date to human-readable string.
toUTCString()	Converts a Date to human-readable string, in Universal time.
parse(datestring)	Returns the number of milliseconds in a date string since 1/1/1970. (datestring: a string containing the date/time to be parsed).
UTC(year, month, [day], [hours], [minutes], [seconds], [milli])	Returns the number of milliseconds in a date string since 1/1/1970, Universal time.
valueOf()	Converts a Date to milliseconds. Same as getTime();.

To write out today's date for example, you would do:

```
<script type="text/javascript">
var mydate= new Date()
var theyear=mydate.getFullYear()
var themonth=mydate.getMonth()+1
var thetoday=mydate.getDate()

document.write("Today's date is: ")
document.write(theyear+"/"+themonth+"/"+thetoday)

</script>
```



Below example returns the day of the week a specific past date falls on:

```
birthday = new Date(1978,2,11)
weekday = birthday.getDay()
alert(weekday) //alerts 6, or Saturday
```

This example calculates the number of days passed since the Year 2000:

```
var today=new Date()
var year2000=new Date(2000,0,1)
var diff=today-year2000 //unit is milliseconds
diff=Math.round(diff/1000/60/60/24) //contains days passed since Year 2000
```

This example sets the Date object to be 3 days into the future:

```
var today=new Date()
today.setDate(today.getDate()+3) //today now is set to be 3 days into the future
```

JavaScript String Object

The String object of JavaScript allows you to perform manipulations on a stored piece of text, such as extracting a substring, searching for the occurrence of a certain character within it etc. For example:

```
var sitenam="JavaScript"  //example string
alert(sitename.length)    //alerts 10, the total number of characters
```

Methods	Description
anchor(name)	Returns the string with the tag <A name="name"> surrounding it.
big()	Returns the string with the tag <BIG> surrounding it.
blink()	Returns the string with the tag <BLINK> surrounding it.
bold()	Returns the string with the tag <B> surrounding it.
fixed()	Returns the string with the tag <TT> surrounding it.
fontcolor(color)	Returns the string with the tag <FONT color="color"> surrounding it. Example(s).
fontsize(size)	Returns the string with the tag <FONT size="size"> surrounding it.
italics()	Returns the string with the tag <I> surrounding it.
link(url)	Returns the string with the tag <A href="url"> surrounding it.
small()	Returns the string with the tag <SMALL> surrounding it.
strike()	Returns the string with the tag <STRIKE> surrounding it.
sub()	Returns the string with the tag <SUB> surrounding it.
sup()	Returns the string with the tag <SUP> surrounding it.
charAt(x)	Returns the character at the "x" position within the string.
charCodeAt(x)	Returns the Unicode value of the character at position "x" within the string.
concat(v1, v2,...)	Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.
fromCharCode(c1, c2,...)	Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance. For example: String.fromCharCode().
indexOf(substr, [start])	Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is 0. Example(s)
lastIndexOf(substr, [start])	Searches and (if found) returns the index number of the searched character or substring within the string. Searches the string from end to beginning. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is string.length-1.
match(regexp)	Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match is found.
replace(regexp, replacetext)	Searches and replaces the regular expression portion (match) with the replaced text instead.
search(regexp)	Tests for a match in a string. It returns the index of the match, or -1 if not found.
slice(start, [end])	Returns a substring of the string based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.
split(delimiter, [limit])	Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return. Example(s)
substr(start, [length])	Returns the characters in a string beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed. Example(s)
substring(from, [to])	Returns the characters in a string between "from" and "to" indexes, NOT including "to" inself. "To" is optional, and if omitted, up to the end of the string is assumed. Example(s)
toLowerCase()	Returns the string with all of its characters converted to lowercase.
toUpperCase()	Returns the string with all of its characters converted to uppercase.

Examples

fontcolor(color)

```
var myname="Peter Don"
document.write(myname.fontcolor("blue")) //writes out "Peter Don" in blue
```

concat(v1, v2,...)

```
var he="Bob"
var she="Jane"
var final=he.concat(" loves ", she) // "Bob loves Jane"
```

indexOf(substr, [start])

```
var message="abcdefg"
message.indexOf("h") //returns -1, as "h" isn't in string

var myname="John Miller"
myname.indexOf("Miller") //returns 5, the starting index of "Miller"
```

split(delimiter, [limit])

```
var sitename="Welcome to JavaScript"
var words=sitename.split(" ") //split using blank space as delimiter
for (i=0; i<words.length; i++)
alert(words[i]) //alerts "Welcome", "to", "JavaScript", and "Kit"
```

substr(start, [length])

```
var sitename="Welcome to JavaScript"
alert(sitename.substr(3, 3)) //alerts "com"
```

substring(from, [to])

```
var sitename="Welcome to JavaScript"
alert(sitename.substring(0, 2)) //alerts "We"
```